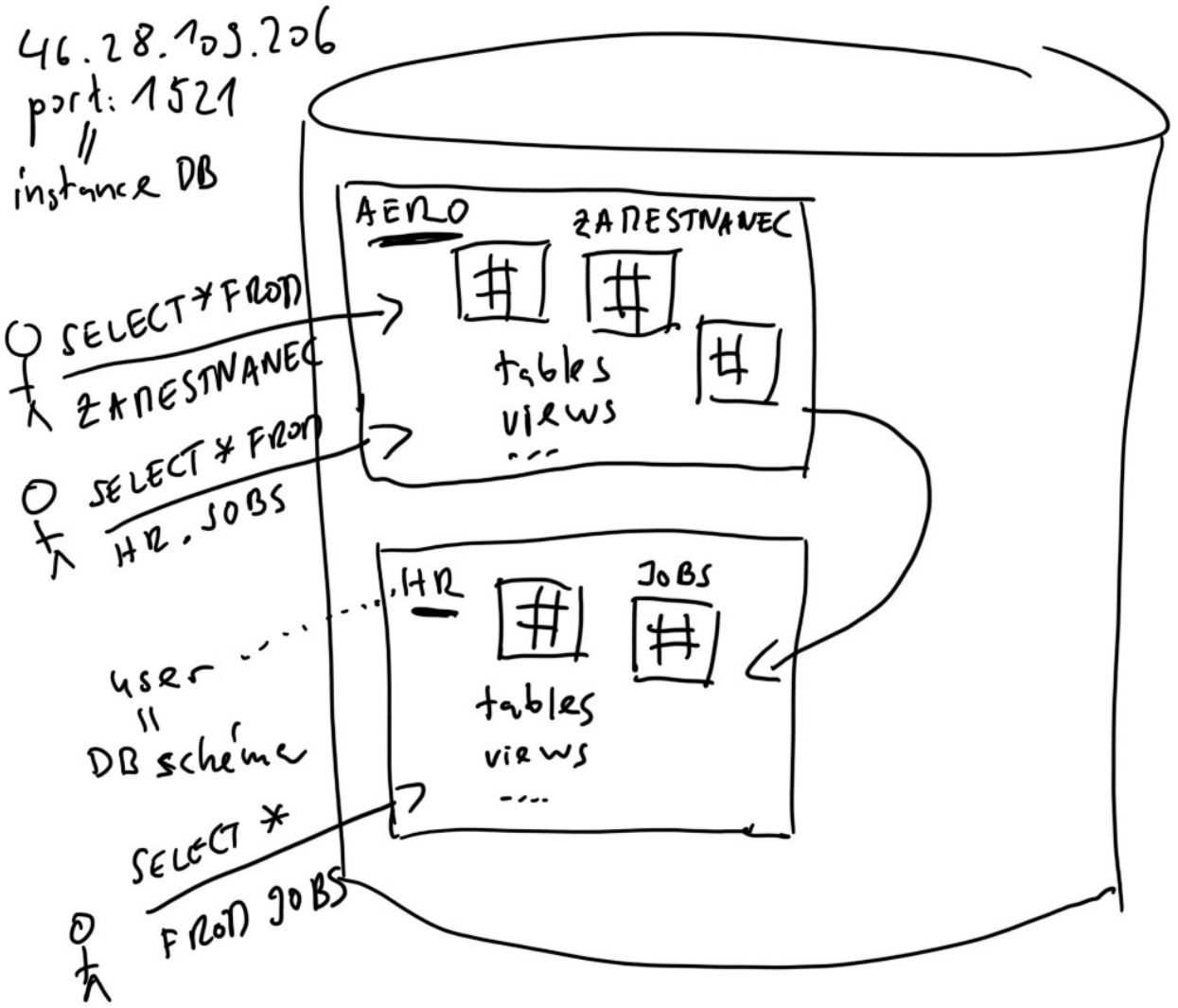
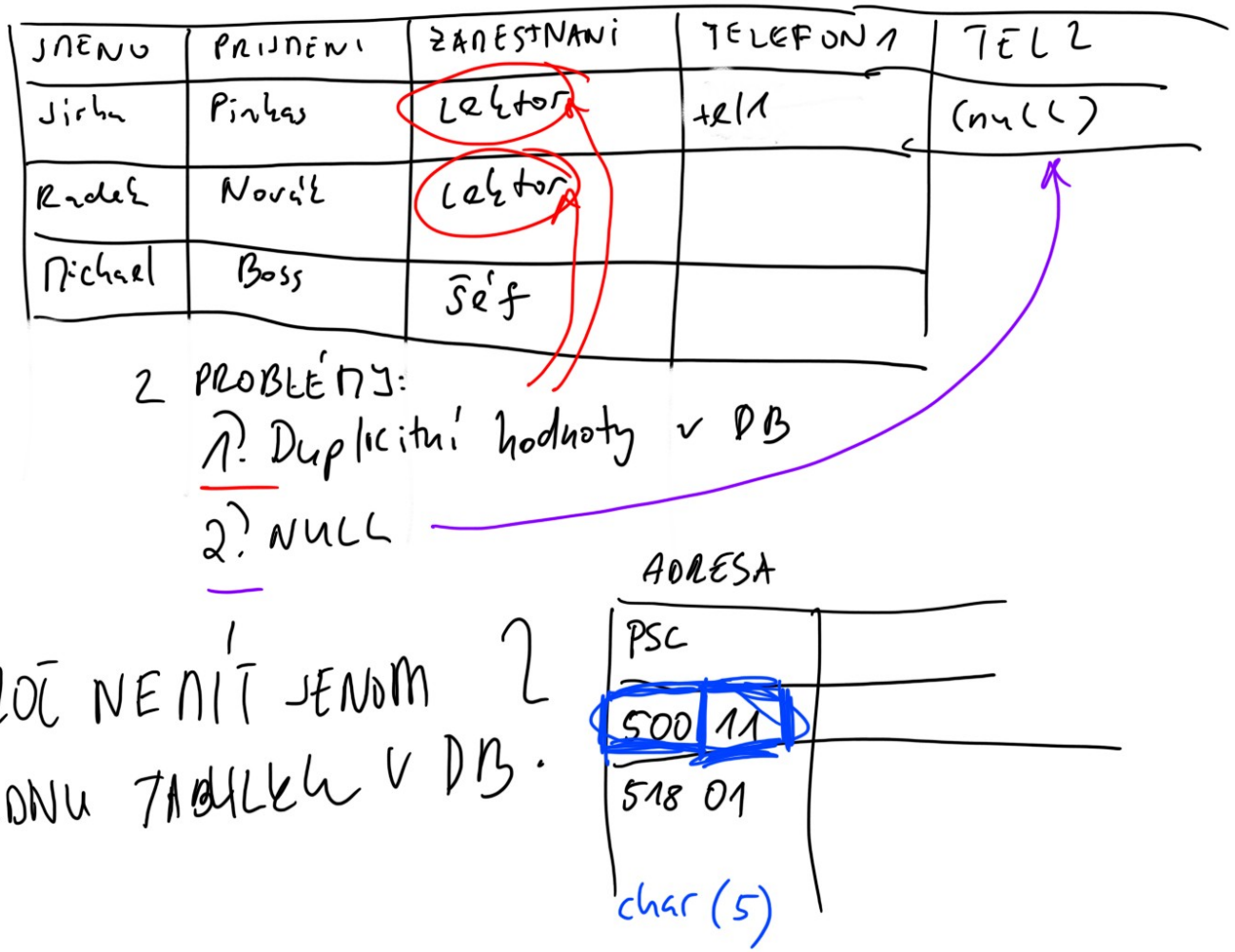
SELECT

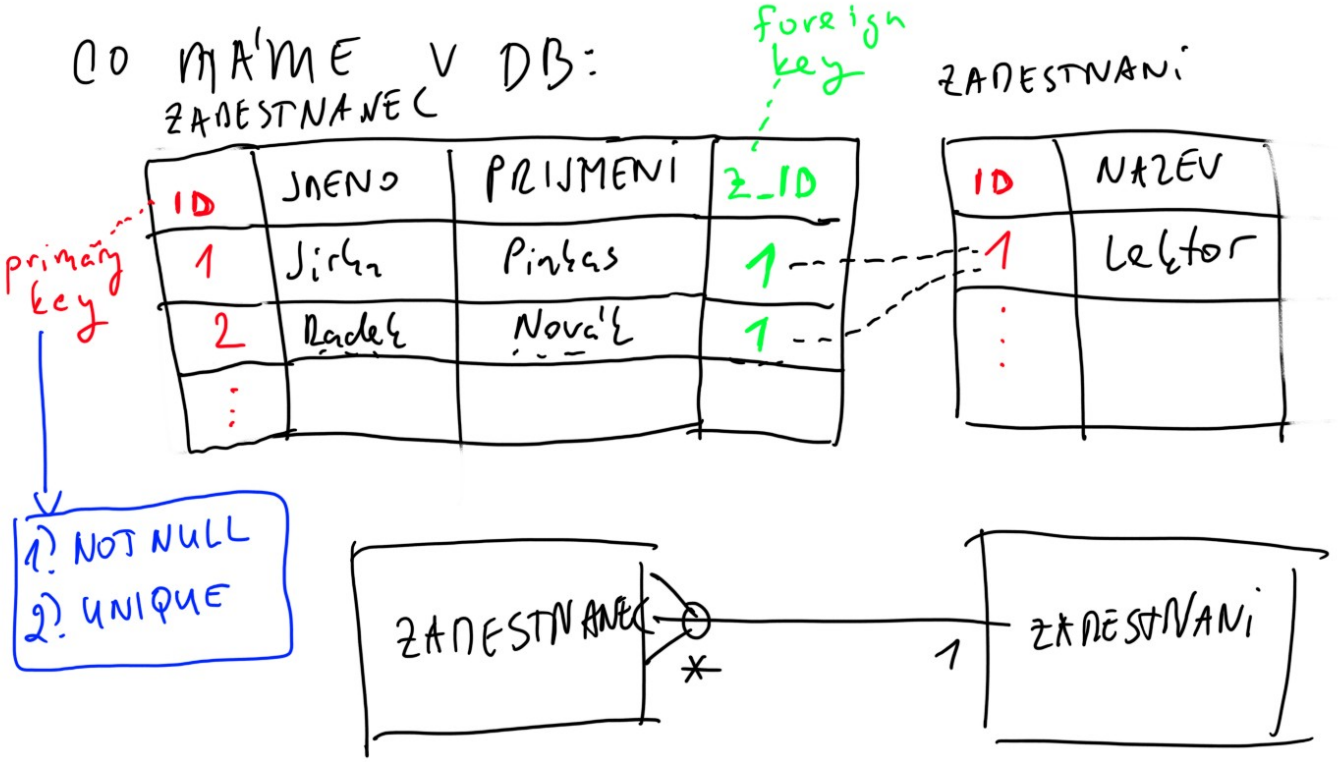
Když se připojíme jako uživatel AERO a máme oprávnění na čtení dat z jiného schématu (HR):



Důvod, proč v databázi nemáme jenom jednu tabulku, ale více tabulek:



Když máme v databázi více tabulek, tak se pro jejich propojení používají primární a cizí klíče:



při návrhu tabulek se používá tzv. "normalizace", správná DB je v tzv. 3. normální formě (3NF)

Příkaz SELECT

- Výběr dat z databázových tabulek – základní syntaxe:

```
<příkaz_select> =
  SELECT <seznam výstupních sloupců>
  FROM <seznam tabulek>
  [WHERE <podmínka pro výběr řádků>]
  [GROUP BY <seznam sloupců
    podle kt. se seskupuje>]
  [HAVING <podmínka pro seskupení>]
  [{UNION|UNION ALL|INTERSECT|MINUS}
    <příkaz_select>]
  [ORDER BY <seznam řazených sloupců>]
```

- Příkaz SELECT má mnoho variant, aby byl schopen vyhledávat data podle nejrůznějších kritérií.
- Příkaz SELECT má povinnou klauzuli FROM a nepovinné klauzule WHERE, GROUP BY, HAVING, ORDER BY.

Příkaz SELECT – určení výstupních sloupců (projekce)

- Výpis všech sloupců (a řádků) v tabulce:
  - `SELECT * FROM zamestnanec;`
- Určení sloupců, které nás zajímají:
  - `SELECT prijmeni, jmeno, plat FROM zamestnanec;`
- Určení aliasů (přezdívek, vlastních jmen) pro sloupce, které chceme pojmenovat/přejmenovat ve výstupu dotazu (klíčové slovo AS je nepovinné):
  - `SELECT AVG(plat) AS prumerny_plat FROM zamestnanec;`
  - `SELECT cena * 1.19 AS cenasdph FROM zbozi;`

Projekce II.

- V případě, že je název sloupce ve výsledné matici unikátní, pak ho nemusíme kvalifikovat názvem tabulky:

```
SELECT prijmeni, jmeno, plat FROM zamestnanec;
```

- Nicméně když to uděláte, pak je dotaz rychlejší, protože říkáte Oraclu kde přesně má hledat sloupce:

```
SELECT zamestnanec.prijmeni, zamestnanec.jmeno,
zamestnanec.plat FROM zamestnanec;
```

(ještě lépe s aliasy tabulek, které jsou na následujícím snímku).

Příkaz SELECT – klauzule FROM

- Určuje, z jakých tabulek se budou vybírat data:
  - `SELECT * FROM zamestnanec;`
  - `SELECT * FROM ucitel, uci, student;`
- Tabulkám lze přiřadit aliasy, pomocí kterých se budeme odkazovat na sloupce tabulek. Alias poslouží pro rozlišení stejně pojmenovaných sloupců v různých tabulkách:
  - `SELECT u.jmeno, s.jmeno FROM ucitel u, uci uc, student s;`
- Pokud má uživatel přístup k více schématům (prostorům s tabulkami), je vhodné před názvem tabulky pro jednoznačnost uvádět i název schématu. Názvem schématu rozlišíme stejně pojmenované tabulky z různých schémat:
  - `SELECT z.jmeno FROM crm.zakaznik z;`

Příkaz SELECT – klauzule WHERE

- Obsahuje podmínku, pomocí které lze omezit počet řádků vrácených příkazem SELECT. Podmínka může obsahovat různá omezení (restrikce) kladená na hodnoty ve sloupcích, poté co by dotaz vrátil kartézský součin řádků tabulek.
- Ve výsledku se zobrazí pouze řádky, které vyhovují podmínce.
- Podmínka se může skládat z více logických výrazů, které vrací true/false, výrazy lze řetězit pomocí operátorů AND, OR, NOT. Ve výrazech se mohou používat operátory <, >, <=, >=, =, <> aj.
- Př.:

```
SELECT z.prijmeni, z.plat FROM zamestnanec z
WHERE z.plat > 20000;
```

```
PRIJMENI PLAT
-------- -----
Velický  24000
Režná    23000
```

WHERE

- Tyto dva SELECTy jsou stejné:
- select *
- from retired_artifacts
- where source_group_id = 'javax.servlet' and source_artifact_id = 'jsp-api';
-
- select *
- from retired_artifacts
- where (source_group_id, source_artifact_id) = ('javax.servlet', 'jsp-api');

Operátory

- Umožňují vytvářet složitější výrazy.
- Výraz se může vyskytovat v SQL dotazech na mnoha místech: V podmínce, seznamu sloupců vybíraných z databáze, kdekoliv se může vyskytnout konstantní hodnota...
- Přehled operátorů:
  - **Aritmetické**: + (unární a binární), - (unární a binární), *, /,
  - **relační** (pro porovnávání): <, >, <=, >=, =, <>, [NOT] BETWEEN x AND y, x [NOT] LIKE y [ESCAPE 'z'], [NOT] IN, ANY (nebo SOME), ALL, EXISTS, IS [NOT] NULL,
  - **logické**: AND, OR, NOT,
  - **množinové**: UNION, UNION ALL, INTERSECT, MINUS (nebo EXCEPT),
  - **spojování řetězců**: || (pro Oracle, jiné DB mohou mít +), např.: prijmeni || ', ' || jmeno.
  - **závorky**: (<výraz>).

Aritmetické operátory

- Slouží pro práci s číselnými hodnotami, výsledkem je opět číselná hodnota:
  - **Unární + a -** slouží k vytvoření pozitivní/negativní hodnoty (např.: -(27 + 28), -naklady > 500),
  - **binární +, -, *, /** slouží ke sčítání, odčítání, násobení, dělení (výsledkem dělení je vždy reálné číslo, neslouží k celočíselnému dělení).
- Binární + a – lze použít také pro práci s datem a časem:
  - **Operátor +** umožňuje přičíst k datu zadaný počet dní,
  - **operátor -** umožňuje odečíst od data zadaný počet dní, nebo získat rozdíl mezi dvěma daty v počtu dní.

Relační operátory

- Používají se k porovnávání dvou hodnot, zejména v podmínkách SQL příkazů.
- Výsledkem porovnání je logická hodnota true nebo false. Pokud se na jedné straně (nebo obou stranách) operátoru vyskytne hodnota NULL, výsledkem je NULL.
- Při řazení řetězců (větší, menší, rovno) se používá lexikografické porovnání – jednotlivé znaky se porovnávají podle abecedy. Databázový systém porovnává znaky podle jazyka nastaveného v národnostním (místním) nastavení operačního systému, nebo je potřeba nastavit kódovací stránku explicitně – databázový systém má vlastní nastavení.

Relační operátory

- **Operátory <, >, <=, >=, = (rovno), <> (nerovno)** lze aplikovat na čísla, datum a čas nebo na řetězce.
- **z [NOT] BETWEEN x AND y** – to samé jako [NOT] (z >= x AND z <= y).
- **x [NOT] LIKE y [ESCAPE 'z']** – porovnání řetězce znaků x s maskou y obsahující zástupné znaky (kde % nahrazuje 0 a více znaků a _ nahrazuje právě jeden znak v Oracle). Nepovinné ESCAPE umožňuje určit znak, jehož předřazením před speciální znak (%, _) v masce se zruší speciální význam znaku.
- **[NOT] IN** – vrací true, pokud je [není] hodnota nalevo obsažena v množině napravo. Př.: den IN ('pondeli', 'patek').
- **ANY (nebo SOME)** – vrací true, pokud je výraz určený kombinací <, >, <=, >=, =, <> a ANY (seznam výrazů) pravdivý alespoň pro jednu položku v seznamu výrazů. Např.: vek = ANY (SELECT vek FROM jubileum) vrací true, pokud je hodnota alespoň jednoho výrazu vráceného vnořeným SELECTem rovna hodnotě proměnné vek.

Operátor IN

- Select záznamů, které mají kombinaci hodnot:
  - select * from retired_artifacts where (source_group_id, source_artifact_id) in
  - (
  - ('org.axonframework.com.lmax', 'disruptor'),
  - ('net.sf.dozer', 'dozer'),
  -)

Relační operátory

- **ALL** – vrací true, pokud je výraz určený kombinací <, >, <=, >=, =, <> a ALL (seznam výrazů) pravdivý pro všechny položky v seznamu výrazů. Např.: vek >= ALL (SELECT vek FROM zamestnanec) vrací true, pokud je hodnota proměnné věk >= všem hodnotám, které vrací vnořený SELECT.
- **EXISTS** – vrací true, pokud je vnořeným dotazem, který následuje za EXISTS, vrácen alespoň jeden řádek. Např.: SELECT * FROM oddeleni WHERE EXISTS (SELECT * FROM zamestnanec WHERE zamestnanec.oddeleni_id = oddeleni.oddeleni_id);
- **IS [NOT] NULL** – vrací true, pokud je hodnota nalevo rovna [nerovna] NULL. Např.: SELECT jmeno FROM zakaznik WHERE cislo_platebni_karty IS NULL;

Logické operátory

- Slouží pro práci s logickými hodnotami TRUE, FALSE a hodnotami NULL. Vstupem jsou logické hodnoty (nebo NULL), výsledkem je opět logická hodnota (nebo NULL).
  - **AND** – vrací TRUE, pokud jsou oba operandy TRUE.
  - **OR** – vrací TRUE, pokud je alespoň jeden operand TRUE.
  - **NOT** – vrací TRUE, pokud je jediný operand na pravé straně FALSE.
- Kompletní TRUTH tabulky jsou na další straně (všechny kombinace hodnot TRUE, FALSE a NULL). Když je logický operátor aplikován na NULL, pak je výsledek UNKNOWN. UNKNOWN se chová podobně jako FALSE, jediný rozdíl je v tom, že NOT FALSE je TRUE, zatímco NOT UNKNOWN je UNKNOWN.

Truth tabulky I.

- Truth tabulka pro AND:

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

- Truth tabulka pro OR:

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Truth tabulky II.

- Truth tabulka pro NOT:

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Příkaz SELECT – klauzule ORDER BY I.

- Umožňuje zadat sloupce, podle kterých se mají vrácené řádky seřadit vzestupně nebo sestupně. Uvedeno může být více sloupců, podle kterých se bude řadit (např. seřazení vzestupně podle názvů oddělení, poté sestupně podle platů).
- Pokud se řazení neurčí klauzulí ORDER BY, řádky mohou být na výstupu v libovolném nepředvídatelném pořadí.
- Řazení vzestupně/sestupně je určováno klíčovými slovy ASC/DESC, ASC je výchozí možnost, proto se klíčové slovo ASC nemusí u sloupce uvádět.
- Př.:

```
SELECT z.prijmeni, z.plat, o.nazev AS
nazev_oddeleni FROM zamestnanec z, oddeleni o
WHERE z.oddeleni_id = o.oddeleni_id AND z.plat > 20000
ORDER BY o.nazev ASC, z.plat DESC;
```

```
PRIJMENI PLAT  NAZEV_ODDELENI
-------- ----- --------------
Velický  24000 Finanční
Režná    23000 Vývoj
```

Příkaz SELECT – klauzule ORDER BY II.

- Jak se řadí NULL hodnoty?
  - Při vzestupném řazení jsou NULL hodnoty na konci výsledné matice.
  - Při sestupném řazení jsou NULL hodnoty na začátku výsledné matice.
- Toto výchozí nastavení můžete změnit pomocí klíčových slov NULLS FIRST nebo NULLS LAST:

```
SELECT jmeno,
  prijmeni,
  datum_nastupu,
  datum_ukonceni
FROM zamestnanec
ORDER BY datum_ukonceni NULLS FIRST;
```

Příkaz SELECT – klauzule ORDER BY III.

- Také je možné v ORDER BY použít index sloupce. Tento příklad:

```
SELECT z.prijmeni, z.plat
FROM zamestnanec z
ORDER BY o.prijmeni ASC, z.plat DESC;
```

- Je možné přepsat takto:

```
SELECT z.prijmeni, z.plat
FROM zamestnanec z
ORDER BY 1 ASC, 2 DESC;
```

- Osobně tento způsob preferuji pouze, když ve výsledku vracím jenom jeden sloupec a chci podle něj celou výslednou matici utřídit.

Agregační funkce I.

- Pro některé sloupce tabulek (obzvláště těch rozsáhlejších) můžeme chtít získat min./max./průměrnou, ... hodnotu ze všech hodnot v tabulce (tj. pro celou tabulku).
- K těmto výpočtům slouží agregační funkce: MIN (minimum), MAX (maximum), AVG (průměr), COUNT (počet), SUM (součet).
- Př.:

```
SELECT MIN(plat), MAX(plat), AVG(plat),
COUNT(plat), SUM(plat) FROM zamestnanec;
```

```
ZAMESTNANEC_ID PRIJMENI PLAT  ODDELENI_ID
-------------- -------- ----- -----------
             1 Velický  24000           1
             2 Režná    23000           2
```

```
MIN(PLAT) MAX(PLAT) AVG(PLAT) COUNT(PLAT) SUM(PLAT)
--------- --------- --------- ----------- ---------
    23000     24000     23500           2     47000
```

Agregační funkce II.

- Agregační funkce neberou v úvahu řádky s NULL hodnotami, kromě COUNT(*) a GROUPING funkcí
  - Kdybyste je i přesto potřebovali počítat, pak je možné toto omezení obejít pomocí NVL funkce, která nahradí NULL hodnotu za něco jiného. NVL funkce se bude probírat později.
- Většina funkcí může být aplikována na ALL (všechny) nebo DISTINCT (jedinečné) hodnoty, defaultní nastavení je ALL.
  - Tyto dva SELECTy jsou stejné:
    - `SELECT COUNT(datum_nastupu) FROM zamestnanec;`
    - `SELECT COUNT(ALL datum_nastupu) FROM zamestnanec;`
  - Zato tento SELECT spočítá počet dat nástupů zaměstnanců bez duplicitních záznamů:
    - `SELECT COUNT(DISTINCT datum_nastupu) FROM zamestnanec;`

Agregační funkce III.

- Existují i další agregační funkce, které použijete zejména při různých statistických výpočtech.
- Všechny agregační funkce:
  - https://docs.oracle.com/database/121/SQLRF/functions003.htm#SQLRF20035

Tabulka dual

- Klauzule FROM je v databázovém systému Oracle povinná, musí v ní být uvedena alespoň jedna tabulka.
- Pokud je potřeba vypsat na výstup nějaký výraz/výrazy, které nečerpají hodnotu z žádné tabulky, lze použít „fiktivní“ tabulku **dual**, která je předdefinovaná v databázovém systému Oracle ve schématu SYS (jedná se o pomocnou tabulku s jedním řádkem).
- Příklad:

```
SELECT SYSDATE AS aktualni_datum,
       1 + (2 * 34) AS vysledek
FROM dual;
```

- V jiných databázích (MySQL, MSSQL, PostgreSQL) je klíčové slovo FROM nepovinné a tabulka DUAL se tam nenachází. V SAP HANA je tabulka DUMMY, která plní stejnou roli jako tabulka DUAL v Oracle.

DYNAMIČTĚJŠÍ SKRIPTY

Dynamičtější skripty I.: konstanty

- Definice konstanty, pozor na to, že je nutné spouštět jako skript nebo nejprve spustit definici konstanty a poté příslušný select:

```
define velikost = 100;


SELECT TYP_LETADLA.POCET_MIST
FROM TYP_LETADLA
WHERE TYP_LETADLA.POCET_MIST > '&velikost'
```

- **Poznámka:** Když proměnná není definovaná, tak se SQL Developer zeptá na její hodnotu.

Dynamičtější skripty II.: proměnné

- Tento skript se zeptá na hodnotu proměnných za běhu:

```
INSERT INTO PILOT
(ZAMESTNANEC_ID, HODNOST) VALUES
(:zam_id, :hodnost);
```

Spojování tabulek pomocí vnitřního spojení

Spojování tabulek

- Pomocí spojování může příkaz SELECT vybírat data z více tabulek.
- Základní způsob spojení je vyjmenovat v příkazu SELECT v klauzuli FROM více tabulek:
  - `SELECT * FROM zamestnanec, zamestnani;`
- Pokud není uvedena klauzule WHERE je výsledkem kartézský součin tabulek – sada, v níž budou všechny kombinace řádků z obou (případně ze všech) tabulek. Pomocí klauzule WHERE se omezí výsledná sestava jen na řádky, které spolu logicky souvisí (vnitřní spojení tabulek):
  - `SELECT * FROM zamestnanec, zamestnani WHERE zamestnanec.zamestnani_id = zamestnani.zamestnani_id;`

Vnitřní spojení – INNER JOIN I.

- Novější způsob zápisu vnitřního spojení je pomocí klauzule INNER JOIN:
  - ```
    SELECT jmeno, prijmeni, nazev_pozice
    FROM zamestnanec zc, zamestnani zi
    WHERE zc.zamestnani_id = zi.zamestnani_id;
    ```

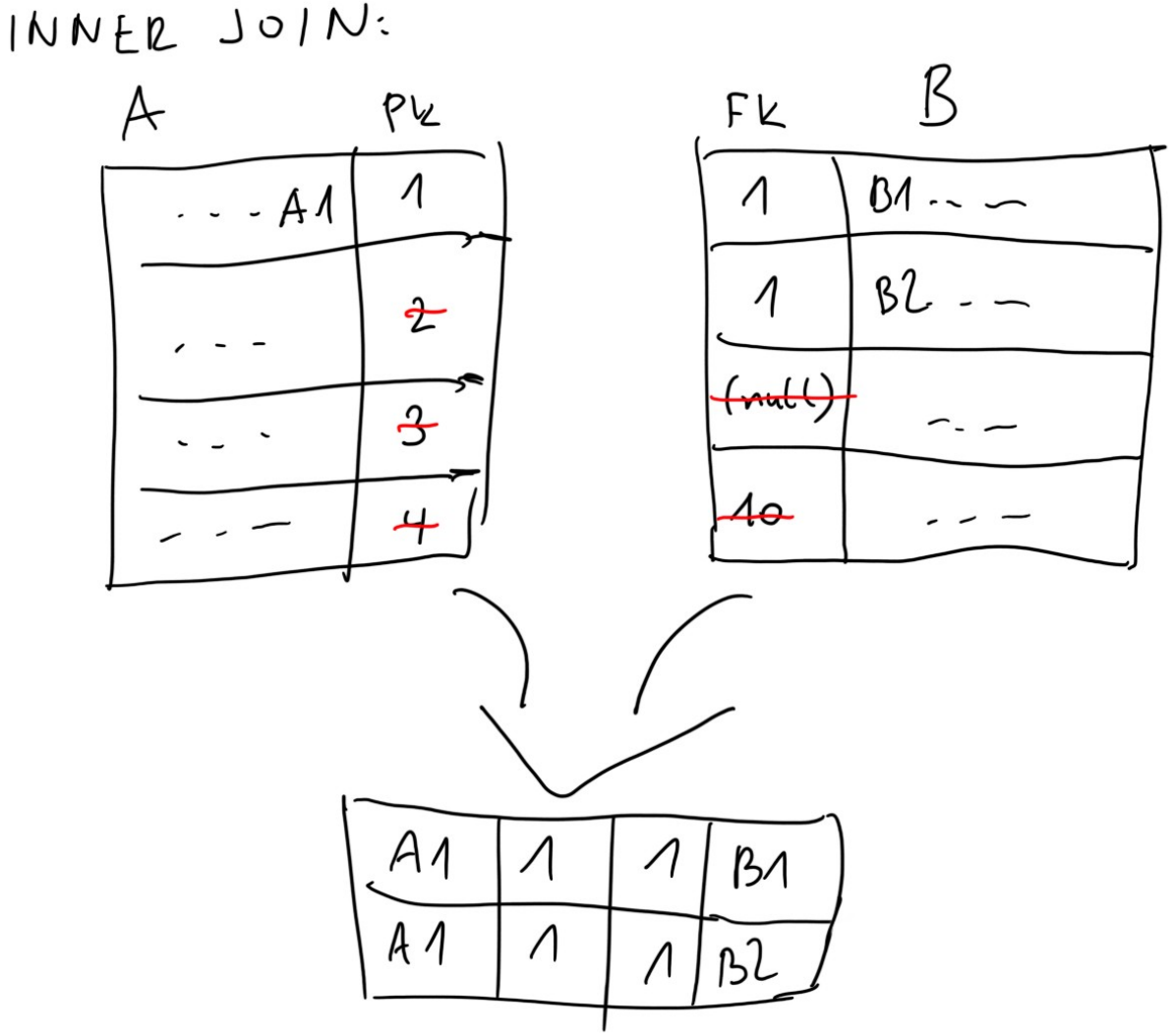
    NEBO:
  - ```
    SELECT jmeno, prijmeni, nazev_pozice
    FROM zamestnanec zc INNER JOIN zamestnani zi
    ON zc.zamestnani_id = zi.zamestnani_id;
    ```

    NEBO:
  - ```
    SELECT jmeno, prijmeni, nazev_pozice
    FROM zamestnanec zc INNER JOIN zamestnani zi
    USING (zamestnani_id);
    ```

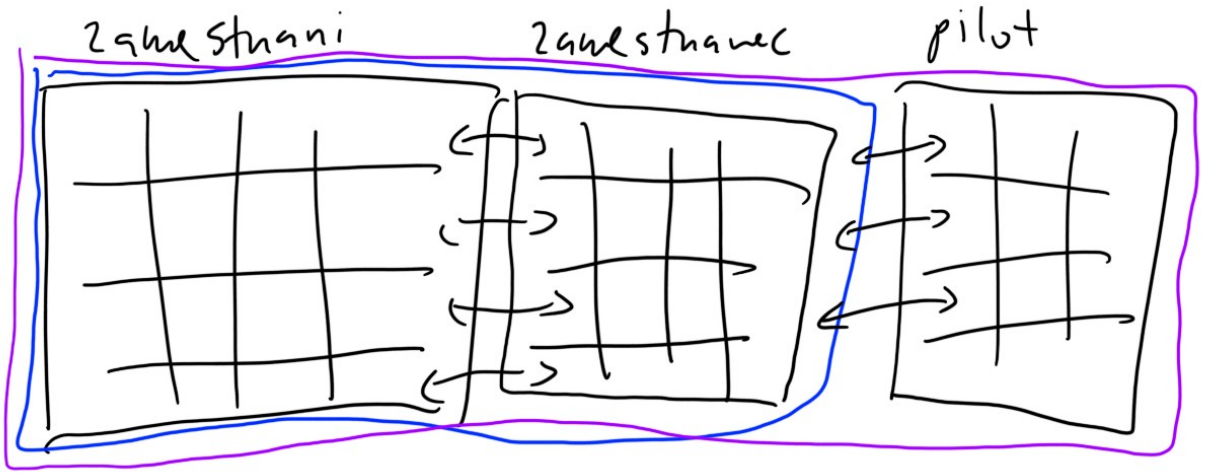
    ← Když se sloupce z tabulky „vlevo“ i z tabulky „vpravo“ jmenují stejně. Pozor: ve výsledné matici bude zamestnani_id jenom jednou a ztratí informaci, z jaké tabulky pochází!

- POZOR: Řádky, které nemají odpovídající záznamy v jiných tabulkách, se do výsledné souhrnné tabulky (sestavy) vůbec nedostanou! (toto řeší vnější spojení, ke kterému se dostaneme později).
- Poznámka: Klíčové slovo INNER je nepovinné.
    ```

Do výsledné matice se u INNER JOIN dostanou pouze ty záznamy, které se „napárují“ zleva i zprava:



Když se spojuje větší množství tabulek, tak je důležité si uvědomit, že se poslední tabulka připojuje nikoli k poslední připojené tabulce, ale k celé aktuálně sestavené matici:



zi + zc + p ✓

p + zc + zi ✓

zc + zi + p ! ✓

zc + p + zi ! ✓

zi + p + zc ✗

p + zi + zc ✗

tady není žádná vazba !!!

Vnitřní spojení – INNER JOIN II.

- Nejčastěji se tabulky spojují přes primární a cizí klíče. Můžete ale tabulky spojovat přes jakékoli sloupce. Přitom dbejte následujících pravidel:
  - Je vhodné, aby na sloupcích podle kterých se spojování tabulek provádí, byly nastaveny indexy (více viz. přednáška o indexech).
  - Nemusíte spojovat pouze přes sloupce, které mají stejný datový typ. V případě že jsou hodnoty kompatibilní, Oracle se pokusí provést implicitní konverzi. Tato operace je ale velice pomalá. Pokud to je možné, tak se něčeho takového vyvarujte. Pokud to možné není, proveďte explicitní konverzi (více viz. přednáška o skalárních funkcích).